2600 (STELLA) Programmer's Guide

By Steve Wright
12/03/79

Updated By Darryl May
7/1/88

Television Protocol

(The TV picture according to Atari)

For purpose of STELLA programming, a single television "frame" consists of 262 horizontal lines, and each line is divided by 228 clock counts (3.58MHz). The actual TV picture is drawn line by line from the top down 60 times a second, and actually consists of only a portion of the entire "frame" (see diagram #1). A typical frame will consist of 3 vertical sync (VSYNC) lines to signal the TV set to start a new frame, 37 vertical blank (VBLANK) lines, 192 TV picture lines, and 30 overscan lines. Atari's research has shown that this pattern will work on all types of TV sets. Each scan line starts with 68 clock counts of horizontal blank (not seen on the TV screen) followed by 160 clock counts to fully scan one line of a TV picture. When the electron beam reaches the end of a scan line, it returns to the left side of the screen , waits for the 68 horizontal blank clock counts, and procedes to draw the next line below.

All horizontal timing is taken care of by hardware, but the microprocessor must "manually" control vertical timing to signal the start of the next frame. When the last line of the previous frame is detected, the microprocessor must generate 3 lines of VSYNC, 37 lines of VBLANK, 192 lines of actual TV picture, and 30 lines of overscan. Fortunately both VSYNC and VBLANK can simply be turned on and off at the appropriate times, freeing the microprocessor for other activities during their execution.

TV FRAME

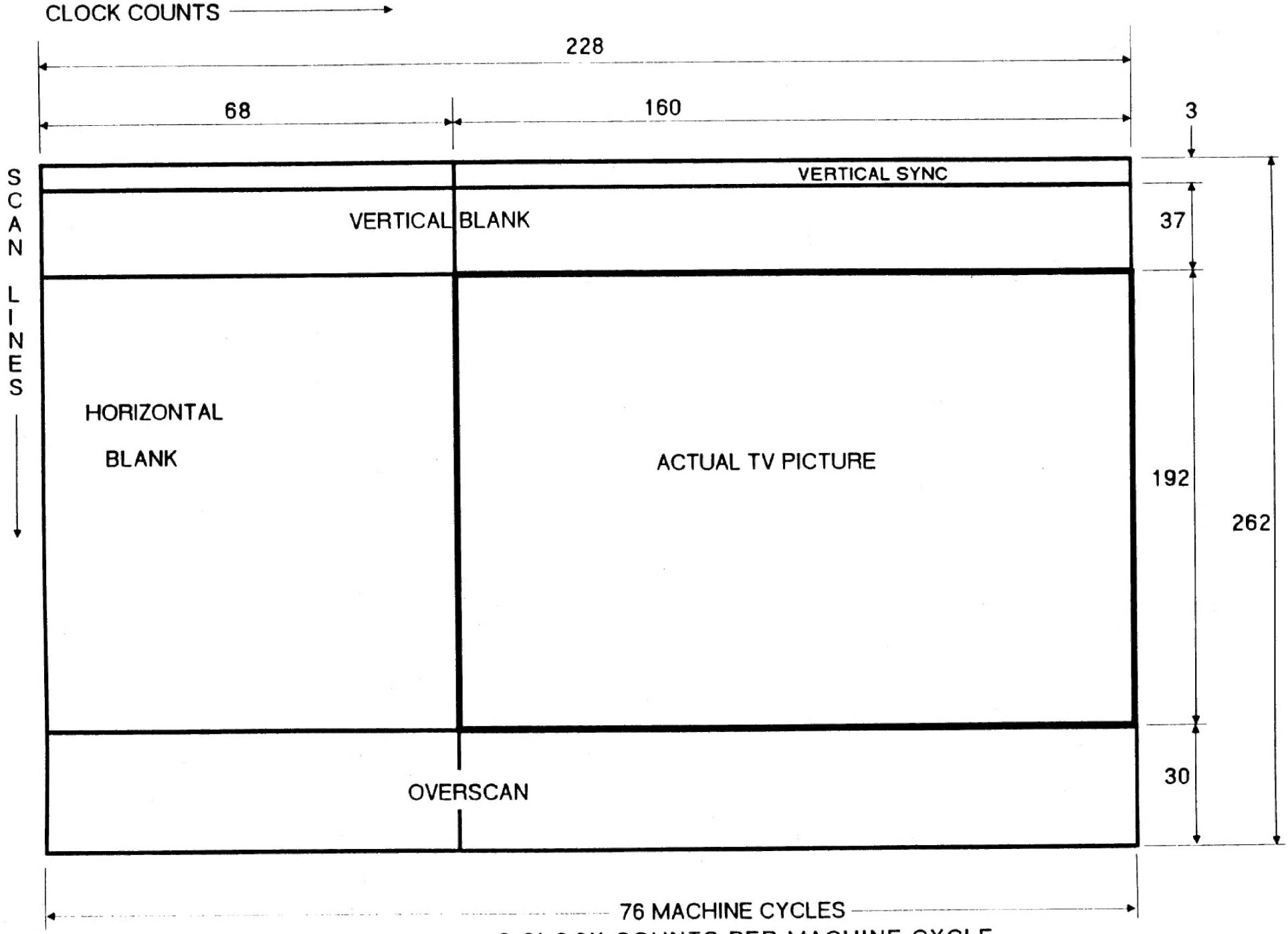

DIAGRAM #1

The actual TV picture is drawn one line at time by having the microprocessor enter the data for that line into the Television Interface Adapter (TIA) chip, which then converts the data into video signals. The TIA can only have data in it that pertains to the line being currently drawn, so the microprocessor must be "one step ahead" of the electron beam on each line. Since one microprocessor machine cycle occurs every 3 clock counts, the programmer has only 76 machine cycles per line (228 / 3 =76) to construct the actual picture (actually less becuase the microprocessor must be ahead of the raster). To allow more time for the software, it is customary (but not required) to update the TIA every two scan lines. The portion of the program that constructs this TV picture is reffered to as the "Kernel", as it is the essence or kernal of the game.

In general, the remaining 70 scan lines (3 for VSYNC, 37 for VBLANK, and 30 for overscan) will provide 5320 machine cycles (70 lines X 76 machine cycles) for houskeeping and game logic. Such activities as calculating the new position of a player, updating the score, and checking for new inputs are typically done during this time.

The TIA

(as seen by the programmer)

1.0 GENERAL DESCRIPTION

The TIA is a custom I.C. designed to create the TV picture and sound from the instructions sent to it by the microprocessor. It converts the 8 bit parallel data from the microprocessor into signals that are sent to the video modulation circuits which combine and shape those signals to be compatable with ordinary TV reception. A "playfield" and 5 moveable objects can be created and manipulated by software.

A playfield consisting of walls, clouds, barriers, and other seldom moved objects can be created over a colored background. The 5 moveable objects can be positioned anywhere, and consist of 2 players, 2 missles, and a ball. The playfield, players, missles, and the ball are created and manipulated by a series of registers in the TIA that the microprocessor can address and write into. Each type of object has certain defined capabilities. For example, a player can be moved with one instruction, but the playfield must be completely re-drawn in order to make it "move".

Color and luminosity (brightness) can be assigned to the background, playfield, and 5 moveable objects. Sound can also be generated and controlled for volume, pitch, and type of sound. Collisions between the various objects on the TV screen are detected by the TIA and can be read by the microprocessor. Input ports which can be read by the microprossor give the status of some of the various hand held controllers.

2.0 THE REGISTERS

All instructions to the TIA are achieved by addressing and writing to various registers in the chip. A key point to remember is that data written in a register is latched and retained until altered by another write operation into that register. For example, if the color register for a player is set for red, that player will be red every time it's drawn until that color register is changed. All of the registers are addressed by the microprocessor as part of the overall RAM/ROM memory space.

All registers have fixed address locations that are pre-assigned address names for handy reference. Many registers do not use all 8 data bits, and some registers are used to "strobe" or trigger events. A "strobe" register executes its function the instant it is written to (the data written is ignored). The only registers a microprocessor can read are the collision registers and input port registers. These registers are conveniently arranged so the data bits of interest always appear as data bits 6 or 7 for easy access.

3.1 HORIZONTAL TIMING

When the electron beam scans across the TC screen and reaches the right edge, it must be turned off and moved back to the left edge of the screen to begin the next scan line. The TIA takes care of this automatically, independent of the microprossor. A 3.58 Mz oscillator generates clock pluses called "color clocks" which go into a pulse counter in the TIA. This counter allows 160 color clocks for the beam to reach the right edge, then generates a horizontal sync signal (HSYNC) to return the beam to the left edge.

It also generates the signal to turn the beam off (horizontal blanking) during its return time of 68 color clocks. Total round trip for the electrom beam is 160 + 68 = 228 color clocks. Again, all the horizontal timing is taken care of by the TIA without assistance from the microprocessor.

3.2 MICROPROCESSOR SYNCHRONIZATION

The microprocessor's clock is the 3.58 MHz oscillator divided by 3, so one machine cycle is 3 color clocks. Therefore, one complete scan line of 228 color clocks allows only 76 machine cycles (228 / 3 =76) per scan line. The microprocessor must be synchronized with the TIA on a line-by-line basis, but program loops and branches take unpredictable lenghts of time. To solve this software synchronization problem, the programmer can use the WSYNC(Wait for SYNC) strobe register. Simply writting to WSYNC causes the microprocessor to halt until the electron beam reaches the right edge of the screen, then the microprocessor resumes operation at the beginning of the 68 color clocks for horizontal blanking. Since the TIA latches all instructions until altered by another write operation, it could be updated every 2 or 3 lines. The advantage is the programmer gains more time to execute software, but a price is paid with lower vertical resolution in the graphics.

NOTE: WSYNC and all the following addresses' bit structures are itemized in the TIA hardware manual. The purpose of this document is to make them understandable.

3.3 VERTICAL TIMING

When the electron beam has scanned 262 lines, the TV set must be signaled to blank the beam and position it at the top of the screen to start a new frame. The signal is called vertical sync, and the TIA must transmit this signal for at least 3 scan lines. This is accomplished by writting a "1" in D1 of VSYNC to turn it on, count at least 3 scan lines, then write a "0" to D1 of VSYNC to turn it off.

To physically turn the beam off during its repositioning time, the TV set needs 37 scan lines of vertical blank signal from the TIA. This is accomplished by writting a "1" in D1 of VBLANK to turn it on, count 37 scan lines, then write a "0" to D1 of VBLANK to turn it off. The microprocessor is of course free to execute other software during these vertical timing commands, VSYNC and VBLANK.

4.0 COLOR AND LUMINOSITY

Color and luminosity can be assigned to the background (BK), playfield (PF), ball (BL), player 0 (P0), player 1 (P1), missile 0 (M0), and missile 1 (M1). There are only four color-lum registers for these 7 objects, so the objects are paired to share the same register according to the following list:

color-lum register	objects colored
COLUMP0	P0 (player 0), M0 (missile 0)
COLUMP1	P1 (player 1), M1 (missile 1)
COLUMPF	PF (playfield), BL (ball)
COLUMBK	BK (background)

For example, if the COLUMP0 register is set for light red, both P0 and M0 will be light red when drawn.

A color-lum register is set for both color and luminosity by writing a single 7 bit instruction to that register. Four of the bits select one of the 16 available colors, and the other 3 bits select on of 8 levels of luminosity (brightness). The specific codes required to create specific color and lum are listed in the the "Detailed Address List" of the TIA hardware manual. As with all registes (except the "strobe" registers) the data written to them is latched until altered by another write operation.

5.0 PLAYFIELD

The PF register is used to create a play field of walls, clouds, barriers, etc., that are seldom moved. This low resolution register is written into to draw the left half of the TV screen only. The right half of the screen is drawn by software selection of either a duplication or a reflection (mirror image) of the left half.

The PF register is 20 bits wide, so the 20 bits are written into 3 addresses: PF0, PF1, and PF2. PF0 is only 4 bits wide and constructs the first 4 "bits" of the playfield, starting at the left edge of the TV screen. PF1 constructs the next 8 "bits", and PF2 the last 8 "bits" which end at the center of the screen. The PF register is scanned from left to right and where a "1" is found the PF color is drawn, and where a "0" is found the BK color is drawn. To clear the playfield, obviously zeroes must be written into PF0, PF1, and PF2.

To make the right half of the playfield a duplication or a copy of the left half, a "0" is written to D0 of the CTLPF (control playfield) register. Writing a "1" will cause the reflection to be displayed.

6.0 THE MOVEABLE OBJECTS GRAPHICS

All 5 moveable objects (P0, M0, P1, M1, BL) can be assigned a horizontal location on the screen and moved left or right relative to that location. Vertical positions, however, are treated in an entirely different manner. In principle, these objects appear at whatever scan lines their graphics registers are enabled. For example, let us assume the ball is to be positioned vertically in the center of the screen. The screen has 192 scan lines and we want the ball to be 2 scan lines "thick". The ball graphics would be disabled until scan line 96, enable for 2 scan lines, then disabled for the rest of the frame. Each type of object (players, missiles, and the ball) has its own characteristics and limitations.

6.1 MISSILE GRAPHICS (M0, M1)

The two missile graphics registers will draw a missle on any scan line by writing a "1" to the one bit enable missile registers (ENAM0, ENAM1). Writing a "0" to these registers will disable the graphics. The missiles' left edge is positioned by a horizontal position register, but the right edge is a function of how wide the missile is made. Width of a missile is controlled by writing into

bits D4 and D5 of the number-size registers (NUSIZ0, NUSIZ1). This has the effect of "stretching" the missiles out over 1, 2, 4, or 8 color clock counts (a full scan line is 160 color clocks).

6.2 BALL GRAPHICS (BL)

The ball graphics register works just like the missile registers. Writing a "1" to the enable ball register (ENABL) enables the ball graphics until the register is disabled. The ball can also be "stretched" to widths of 1, 2, 4, or 8 color clock counts by writing to bits D4 and D5 of the CTRLPF register.

The ball can also be vertically delayed one scan line. For example, if the ball graphics were enabled on scan line 95, it could be delayed to not display on the screen until scan line 96 by writing a "1" to D0 of the vertical delay (VDELBL) register. The reason for having a vertical delay capability is because most programs will update the TIA every 2 lines. This confines all vertical movements of objects to 2 scan line "jumps". The use fo vertical delay allows the objects to move one scan line at a time.

6.3 PLAYER GRAPHICS (P0, P1)

The player graphics are the most sophisticated of all the moveable objects. They have all the capabilities of the missiles and ball graphics, plus three more capabilities. Players can take on a "shape" such as a man or an airplane, and the player can be easily flipped over horizontally to display the mirror image (reflection) instead of the original image, plus multiple copies of the players can be created.

The player graphics are drawn line-by-line like all other graphics. The difference here is each scan line of the player is 8 "bits" wide, whereas the missiles and ball are one "bit" wide. Therefore, a player can be thought of as being drawn on graph paper 8 squares wide and as tall as desired. To "color in the squares" of this imaginary graph paper, 8 data bits are written into the players graphics registers (GP0, GP1). This 8 bit register is scanned from D7 to D0, and wherever a "1" is found that "square" gets the players' color (from the color-lum register) and where a "0" is found that "square" gets the background color. To position a player vertically, simply leave all "0's" in the graphics registers (GP0, GP1) until the electron beam is on the scan line desired, write to the graphics register line-by-line describing the player, then write all "0's" to turn off the players' graphics until the end of that frame.

To display a mirror image (reflection) instead of the original figure write a "1" to D3 of the one bit reflection register (REFP0, REFP1). A "0" written to these registers restores the original figure.

Multiple copies of players as well as their size are controlled by writing 3 bits (D0, D1, D2) into the number-size registers (NUSIZ0, NUSIZ1). These three bits select from 1 to 3 copies of the player, spacing of those copies, as well as the size of the player (each "square" of the player can be 1, 2, or 4 clocks wide). Whenever multiple copies are selected, the TIA automatically creates the same number of copies of the missile for

that player. Again, the specifics of all this are laid out in the TIA hardware manual.

Vertical delay for the players works exactly like the ball by writing a "1" to D0 in the players' vertical delay registers (VDELP0, VDELP1). Writing a "0" to these locations disables the vertical delay.

7.0 HORIZONTAL POSITIONING

The horizontal position of each object is set by writing to its' associated reset register (RESP0, RESP1, RESM0, RESM1, RESBL) which are all "strobe" registers (they trigger their function as soon as they are addressed). That causes the object to be positioned wherever the electron beam was in its sweep across the screen when the register was reset. For example, if the electron beam was 60 color clocks into a scan line when RESP0 was written to, player 0 would be positioned 60 color clocks "in" on the next scan line. Whether or not P0 is actually drawn on the screen is a function of the data in the GP0 register, but if it were drawn, it would show up at 60. Resets to these registers anywhere during horizontal blanking will position objects at the left edge of the screen(color clock 0). Since there are 3 color clocks per machine cycle, and it can take up to 5 machine cycles to write to a register, the programmer is confined to positioning the objects at 15 color clock intervals across the screen. This "course" positioning is "fine tuned" by the Horizontal Motion, explained in section 8.0.

Missiles have an additional positioning command. Writing a "1" to D1 of the reset missile-to-player register (RESMP0, RESMP1) disables that missiles' graphics (turns it off) and repositions it horizontally to the center of it's associated player. Until a "0" is written to the register, the missiles' horizontal position is locked to the center of it's player in preparation to be fired again.

8.0 HORIZONTAL MOTION

Horizontal motion allows the programmer to move any of the 5 graphics objects relative to their current horizontal position. Each object has a 4 bit horizontal motion register (HMP0, HMP1, HMM0, HMM1, HMBL) that can be loaded with a value in the range +7 to -8 (negative values are expressed in two's complement form). This motion is not executed until the HMOVE register is written to, at which time all motion registers move their respective objects. Objects can then be move repeatedly by simply executing HMOVE. Any object that is not to move must have "0" in its motion register. With the horizontal positioning command confined to positioning objects at 15 color clock intervals, the motion registers fill in the gaps by moving objects +7 to -8 color clocks. Objects can now be placed at any color clock position across the screen. All 5 motion registers can be set to zero simultaneously by writing to the horizontal motion clear register (HMCLR).

There are timing constraints for the HMOVE command. The HMOVE command must immediatley follow a WSYNC (Wait for SYNC) to insure the HMOVE operation occurs during horizontal blanking. This is to allow sufficient time for the motion registers to do their thing

before the electron beam starts drawing the next scan line. Also, for mysterious internal hardware considerations, the motion registers should not be modified for at least 24 machine cycles after an HMOVE command.

9.0 OBJECT PRIORITIES

Each object is assigned a priority so when any two objects everlap the one with the highest priority will appear to move in front of the other. To simplify hardware logic, the missles have the same priority as their associated player, and the ball has the same priority as the playfield. The background, of course, has the lowest priority. The following table illustrates the normal (default) priority assignments:

PRIORITY	OBJECTS
1	P0, M0
2	P1, M1
3	BL, PF
4	BK

This priority assignment means that players and missiles will move in front of the playfield. To make the players and missiles move behind the playfield, a "1" must be written to D2 of the CTRLPF register.

The following table illustrates how the priorities are affected:

PRIORITY	OBJECTS
1	PF, BL
2	P0, M0
3	P1, M1
4	BK

One more priority control is available to be used for displaying the score. When a "1" is written to D1 of the CTRLPF register, the left half of the playfield takes on the color of player 0, and the right half takes on the color of player 1. The game score can now be displayed using the PF graphics register, and the score will be in the same color as its associated player.

10.0 COLLISIONS

The TIA detects collisions between any of the 6 object it generates (the playfield and 5 moveable objects). There are 15 possible two-object collisions which are stored in 15 one bit latches. Each collision register contains two of these latches which are read by the microprocessor on D6 and D7 of the data bus for easy access. A "1" on the data line indicates one of the 15 collisions has occured. The collision registers could be read at any time but is usually done during vertical blank after all possible collisions have occured. The collision registers are all reset simultaneously by writing to the collision reset register (CXCLR).

11.0 SOUND

There are two audio channels for sound generation. They are identical but completely independent and can be operated simultaneously to produce sound effects through the TV's speaker. Each audio channel has three registers that control a noise-tone generator (what kind of sound), a frequency selection (high or low pitch of the sound), and a volume control.

11.1 NOISE-TONE GENERATOR

The noise-tone generator is controlled by writing to the 4 bit audio control registers (AUDC0, AUDC1). The values written cause different kinds of sounds to be generated. Some are pure tones like a flute, other have various "noise" content like a rocket motor or explosion. Even though the TIA hardware manual list the sounds created by each value, some experimentation will be necessary to find "your sound".

11.2 FREQUENCY CONTROL

Frequency selection is controlled by writing to a 5 bit audio frequency register (AUDF0, AUDF1). The value written is used to divide a 30KHz reference frequency creating higher or lower pitch of whatever type of sound is created by the noise-tone generator. By combining the pure tones available from the noise-tone generator with a frequency selection, a wide range of tones can be generated.

11.3 VOLUME

Volume is controlled by writing to a 4 bit audio volume register (AUDV0, AUDV1). Writing a "0" to these registers turns sound off completely, and writing any value up to 15 increases the volume accordingly.

12.0 INPUT PORTS

They are six input ports whose logic states can be read on D7 by reading the input port addresses (INPT0, INPT1, INPT2, INPT3, INPT4, INPT5). These six ports are divided into two types, "dumped" and "latched".

12.1 DUMPED INPUT PORTS (INPT0, INPT1, INPT2, INPT3)

These four ports are used to read up to four paddle controllers. Each paddle controller contains an adjustable pot controlled by the knob on the controller. The output of the port is used to charge a capicitor in the console, and when the capacitor is charged the input port goes HIGH. The microprocessor discharges this capacitor by writing a "1" to D7 of VBLANK then measures the time it takes to detect a logical one at the port. This information can be used to position objects on the screen based on the position of the knob on the paddle controller.

12.2 LATCHED INPUT PORTS (INPT4, INPT5)

These two ports have latches that are both enabled by wring a "1" or disabled by writing a "0" to D6 of VBLANK. When disabled the microprocessor reads the logic level of the port directly. When enabled, the microprocessor is reading the latch, not the port. When enabled, the latch is set for logic one and remains that way until its' port goes LOW. When the port goes LOW the latch goes LOW and remains that way regardless of what the port does. The trigger buttons of the joystick controllers connect to these ports.

1.0 GENERAL

The PIA chip is an off-the-shelf 6532 Peripheral Interface Adaptor which has three functions, a programmable timer, 128 bytes of RAM, and two 8 bit parallel I/O ports.

2.0 INTERNAL TIMER

The PIA uses the same clock as the microprocessor so that one PIA cycle occurs for each machine cycle. The PIA can be set for one of four different "intervals", where each interval is some multiple of the clock (and therefore machine cycles). A value from 1 to 255 is loaded into the PIA which will be decremented by one at each interval. The timer can now be read by the microprocessor to determine elasped time for timing various software operations and keep them synchronized with the hardware (TIA chip).

2.1 SETTING THE TIMER

The timer is set by writing a value (from 1 to 255) to the address of the desired interval setting according to the following table:

HEX ADDRESS	INTERVAL	MNEMONIC
$294	1 Clock	TIM1T
$295	8 Clocks	TIM8T
$296	64 Clocks	TIM64T
$297	1,024 Clocks	TIM1024T

For example, if the value 100 were written to TIM64T ($296) the timer would decrement to 0 in 6400 clocks (64 clocks per interval x 100 intervals) which would also be 6400 microprocessor machine cycles.

2.2 READING THE TIMER

The timer may be read any number of times after it is loaded of course, but the programmer is usually interested in whether or not the timer has reached 0. The timer is read by reading INTIM at HEX address $284.

2.3 WHEN THE TIMER REACHES ZERO

The PIA decrements the value loaded into it once each interval until it reaches 0. It holds that 0 for one interval, then the value is flipped over to $FF and decrements once each clock cycle, rather than once per interval. The purpose of this feature is to allow the programmer to determine how long ago the timer zeroed itself out in the event the timer was read after it passed zero.

3.0 RAM

The PIA has 128 bytes of RAM located in the STELLA map from HEX address $80 to $FF. The microprocessor stack is normally located from $FF downward, and variables are normally located from $80 upward (hoping the two never meet).

4.0 THE I/O PORTS

The two ports (Port A and Port B) are 8 bits wide and can be set for either input or output. Port A is used to interface to various hand-held controllers but Port B is dedicated to reading the status of the STELLA console switches.

4.1 PORT B - Console Switches (Read only)

Port B is hardwired to be an input port only. Port B is read by addressing SWCHB ($282) to determine the status of all the console switches according to the following table:

DATA BIT	SWITCH	BIT MEANING
D7	P1 Difficulty	0 = Amateur (B), 1 = Pro (A)
D6	P0 Difficulty	0 = Amateur (B), 1 = Pro (A)
D5	Not used.	
D4	Not used.	
D3	Color - B/W	0 = B/W, 1 = Color
D2	Not used.	
D1	SELECT	0 = Switch is pressed.
D0	RESET	0 = Switch is pressed.

NOTE: All the above switches work the same on the 7800 except for the "Color - B/W" switch which is the pause button on the 7800.

5.0 PORT A - Hand Controllers

Port A is under full software control to be configured as an input or an output port. It can then be used to read or control various hand-held controllers with the data bits defined differently depending on the type of controller used.

5.1 SETTING FOR INPUT OR OUTPUT

Port A has an 8 bit wide Data Direction Register (DDR) that is written to at CTLSWA ($281) to set each individual pin of Port A to be either input or output. The Port A pins are labeled PA0 through PA7. Writing a "0" to a pins' DDR bit will set that pin for input and writing a "1" to a pins' DDR bit will set that pin for output. For example, writing all 0's to CTLSWA (the DDR for Port A) sets PA0 to PA7 (all 8 pins of Port A) as inputs. If $F0 (11110000) were written to CTLSWA then PA7, PA6, PA5, & PA4 would be outputs, and PA3, PA2, PA1 & PA0 would be inputs.

5.2 INPUTING AND OUTPUTING

Once the DDR has set the pins of Port A for input or output they may be read or written to by addressing SWCHA ($280).

5.3 JOYSTICK CONTROLLERS

Two joysticks can be read by configuring the entire port as input and reading the data at SWCHA ($280) according to the following table:

DATA BIT	DIRECTION	PLAYER
D7	Right	P0 (Left Player)
D6	Left	P0
D5	Down	P0
D4	Up	P0
D3	Right	P1 (Right Player)
D2	Left	P1
D1	Down	P1
D0	Up	P1

A "0" in a data bit indicates the joystick has been moved to close that switch. All "1"s in a player's "nibble" indicates the joystick is not moving.

NOTE: The trigger buttons do not go to the PIA. They are read on bit 7 of INPT4 and INPT5 of the TIA.

5.4 PADDLE (Pot) CONTROLLERS

Only the paddle triggers are read from the PIA. The paddles themselves are at INPT0 through INPT3 of the TIA. The data bit is set to 0 when the trigger is pressed. The paddle triggers can be read at SWCHA according to the following table:

DATA BIT	PADDLE NUMBER
D7	Paddle 0
D6	Paddle 1
D5	Not used.
D4	Not used.
D3	Paddle 2.
D2	Paddle 3.
D1	Not used.
D0	Not used.

5.5 KEYBOARD CONTROLLERS

The keyboard controller has 12 buttons arranged into 4 rows and 3 columns. A signal is sent to a row, then the columns are checked to see if a button is pushed, then the next row is signaled and all columns are sensed, etc. until the entire keyboard is scanned and sensed. The PIA sends the signals to the row, and the columns are sensed by reading INPT0, INPT1, and INPT4 of the TIA. With Port A configured as an output port, the data bits will send a signal to the keyboard controller rows according to the following table:

DATA BIT	KEYBOARD ROW	PLAYER	
D7	Bottom	P0	(Left Player)
D6	Third	P0	
D5	Second	P0	
D4	Top	P0	
D3	Bottom	P1	(Right Player)
D2	Third	P1	
D1	Second	P1	
D0	Top	P1	

NOTE: A delay of 400 microseconds is necessary between writting to this port and reading the TIA input ports.

6.0 ADDRESS SUMMARY TABLE

HEX ADDRESS	MNEMONIC	PURPOSE
$280	SWCHA	Port A; Input or Ouput (Read or Write)
$281	CTLSWA	Port A DDR, 0=Input, 1=Output (Write only)
$282	SWCHB	Port B; Console switches (Read only)
$283	CTLSWB	Port B DDR (Hardwired as input)
$284	INTIM	Current timer interval count. (Read only)
$294	TIM1T	Set 1 clock interval (838 nsec/interval)
$295	TIM8T	Set 8 clock interval (6.7 usec/interval)
$296	TIM64T	Set 64 clock interval (53.6 usec/interval)
$297	TIM1024	Set 1024 clock interval (858.2 usec/interval)

NOTE: One clock is also one microprocessor machine cycle.

PAL/SECAM CONVERSIONS

PAL

1. The number of scan lines, and therefore the frame time, increases from NTSC to PAL according to the following table:

	NTSC		PAL	
	SCAN LINES	MICRO-SECONDS	SCAN LINES	MICRO-SECONDS
VBLANK	40	2,548	48	3,085
KERNAL	192	12,228	228	14,656
OVERSCAN	30	1,910	36	2,314
FRAME	262	16,686	312	20,055

2. Sounds will drop a little in pitch (frequency) because of a slower crystal clock. Some sounds may need the AUDF0/AUDF1 touched up.

3. PAL operates at 50 Hz compared to NTSC 60 Hz, a 17% reduction. If game play speed is based on frames per second, it will slow down by 17%. This can be disastrous for most skill/action carts. If the NTSC version is designed with 2 byte fractional addition techniques (or anything not based on frames per second) to move objects, then PAL conversion can be as simple as changing the fraction tables, avoiding major surgery on the program.

SECAM

1. SECAM is a little weird. It takes the PAL software, but the console color/black & white switch is hardwired as black & white. Therefore, it reads the PAL black & white tables in software and assigns a fixed color to each lum of black & white according to the following table:

LUM	COLOR
0	black
2	blue
4	red
6	magenta
8	green
A	cyan
C	yellow
E	white

There is a trap here: the manual is the same for NTSC, PAL, & SECAM. This means that the descriptions for black & white must jive between NTSC & PAL. If you make major changes to PAL black & white to achieve good SECAM color, NTSC black & white must be made similar.

2. PAL sounds work fine on SECAM with one exception: when a sound is to be turned off, it must be done by setting AUDV0/AUDV1 to 0, not by setting AUDC0/AUDC1 to 0. Otherwise, you get an obnoxious background sound.